LPI exam 201 prep: Filesystem

Intermediate Level Administration (LPIC-2) topic 203

Skill Level: Intermediate

David Mertz, Ph.D. (mertz@gnosis.cx)
Developer
Gnosis Software

31 Aug 2005

In this tutorial, David Mertz continues preparing you to take the Linux Professional Institute® Intermediate Level Administration (LPIC-2) Exam 201. In this third of eight tutorials, you will learn how to control the mounting and un-mounting of filesystems, examine existing filesystems, create filesystems, and perform remedial actions on damaged filesystems.

Section 1. Before you start

Learn what these tutorials can teach you and how you can get the most from them.

About this series

The Linux Professional Institute (LPI) certifies Linux system administrators at junior and intermediate levels. To attain each level of certification, you must pass two LPI exams.

Each exam covers several topics, and each topic has a weight. The weights indicate the relative importance of each topic. Very roughly, expect more questions on the exam for topics with higher weight. The topics and their weights for LPI exam 201 are:

Topic 201
Linux kernel (weight 5).

Topic 202

System startup (weight 5).

Topic 203
Filesystem (weight 10). The focus of this tutorial.

Topic 204
Hardware (weight 8).

Topic 209
File and service sharing (weight 8).

Topic 211
System maintenance (weight 4).

Topic 213
System customization and automation (weight 3).

Topic 214
Troubleshooting (weight 6).

The Linux Professional Institute does not endorse any third-party exam preparation material or techniques in particular. For details, please contact info@lpi.org.

About this tutorial

Welcome to "Filesystem," the third of eight tutorials designed to prepare you for LPI exam 201. In this tutorial, you will learn how to control the mounting and un-mounting of filesystems, examine existing filesystems, create filesystems, and perform remedial actions on damaged filesystems.

The tutorial is organized according to the LPI objectives for this topic, as follows:

2.203.1 Operating the Linux filesystem (weight 3)
You will be able to properly configure and navigate the standard Linux filesystem. This objective includes configuring and mounting various filesystem types. Also included is manipulating filesystems to adjust for disk space requirements or device additions.

2.203.2 Maintaining a Linux filesystem (weight 4)
You will be able to properly maintain a Linux filesystem using system utilities. This objective includes manipulating a standard ext2 filesystem.

2.203.3 Creating and configuring filesystem options (weight 3)
You will be able to configure automount filesystems. This objective includes configuring automount for network and device filesystems. Also included is creating non-ext2 filesystems for devices such as CD-ROMs.

This tutorial addresses elements of Linux as well as external tools that are useful for working with Linux systems. Support for filesystems, devices, and partitions is either compiled into the base kernel or included in kernel modules.

However, various tools that you are likely to use in managing these filesystems recognized by Linux are userland utilities and therefore only commonly included with Linux distributions rather than part of Linux itself. Nonetheless, filesystem tools are essential for working with pretty much every Linux system regardless of its intended use (even non-networked or embedded systems).

Prerequisites

To get the most from this tutorial, you should already have a basic knowledge of Linux and a working Linux system on which you can practice the commands covered in this tutorial.

Section 2. Creating and configuring filesystem options

Let's go out of order and start with creating and configuring filesystems and options.

Creating partitions

Before you can work with Linux filesystems, you need to create them. But before you can create a filesystem, you need to create a partition to put it on. As a brief primer, on x86 machines, hard disks may be divided into four primary partitions, but the last of those primary partitions may contain a number of extended partitions inside it.

In the past there were a number of restrictions about the highest cylinders where bootable partitions can occur, maximum disk sizes, locations of primary partitions on large disks, and so on. However, for the last five years or more, pretty much all system BIOSes flexibly handle disks of essentially unlimited size, and modern bootloaders (at least for Linux) have no important restrictions about partition sizes or locations.

The only rule that remains to worry about nowadays concerns operating systems other than Linux. Sometimes those still insist on living in primary partitions near the front of a hard disk. Linux partitions are more than happy to reside on extended partitions and anywhere on any accessible disk drive.

There are several widely used tools in the Linux world for creating and manipulating partitions on hard disks. The oldest such tool is *fdisk*. Somewhat later, the *curses*-based *cfdisk* became popular. GNU *parted* is also used in many distributions. Further, the installation systems for most Linux distributions and/or their graphical environments come with partitioning front-ends that provide friendlier interfaces to viewing and modifying partitions.

Of these tools, fdisk remains the most flexible and most forgiving tool. But "forgiving"

is a slightly odd term to use here. Writing unintended partition-table information is a recipe for disaster regardless of what tool you use. But if your partitions have been created in somewhat non-standard ways, often by non-Linux operating systems and tools, fdisk will generally forge ahead where other tools might refuse to try at all. If it works, however, cfdisk is generally friendlier and more interactive. And parted provides more powerful options about resizing and moving existing partitions non-destructively than fdisk or cfdisk.

Whatever tool you use to create partitions, the concepts are similar. First, you need to perform these operations as root, ideally in single-user mode. And it's hard to make this point too strongly: *Be careful* when you modify partitions: ideally, have all important data backed up, and pay careful attention to what changes you make.

Before you start modifying a partition table, it is a good idea to be clear about what partitions currently exist. The command `fdisk -l /dev/hda` (or similar for other disks, for example /dev/hdb or /dev/sda) gives you information on existing partitions. `mount` is also helpful in understanding how these existing partitions are actually being used. If you wish to create new partitions, keep in mind any extra sectors within the fourth primary partition that might be available for additional extended partitions.

Let's see an example of a partition table on a Linux system of mine:

Listing 1. Sample partition table

```
% fdisk -l /dev/sda
Disk /dev/sda: 80.0 GB, 80026361856 bytes
255 heads, 63 sectors/track, 9729 cylinders

Device Boot   Start      End      Blocks     Id  System
/dev/sda1   *        1     1216     9767488+     7  HPFS/NTFS
/dev/sda3         1217     4255    24410767+    83  Linux
/dev/sda4         4256     9729    43969905      5  Extended
/dev/sda5         4256     4380     1004031     82  Linux swap /
Solaris
/dev/sda6         4381     5597     9775521     83  Linux
```

This tells us several things. First of all, we can see that partition one is probably used by a foreign operating system. And running `mount` will let us know:

```
% mount | head -1
/dev/sda3 on / type reiserfs (rw,noatime,notail,commit=600)
```

That is, the existing system is rooted on /dev/sda3. Perhaps most interestingly, the /dev/sda4 partition extends to cylinder 9729, but the extended partitions within it use only part of that space.

After discovering some free space available on the drive, let's create a partition within it using fdisk:

```
% fdisk /dev/sda
```

The number of cylinders for this disk is set to 9729. There is nothing wrong with that,

but this is larger than 1024 and could, in certain setups, cause problems with:

1. Software that runs at boot time (such as old versions of LILO).
2. Booting ant partitioning software from other operating systems (such as DOS FDISK, OS/2 FDISK).

Listing 2. Creating a partition

```
Command (m for help): n
Command action
l   logical (5 or over)
p   primary partition (1-4)
l
First cylinder (5598-9729), default 5598):
Using default value 5598
Last cylinder or +size or +sizeM or +sizeK (5598-9729, default 9729):
+10000M

Command (m for help): w
The partition table has been altered!
```

Everything that follows a colon is typed in by the user (you). At this point, we have created a new 10 GB Linux partition:

```
/dev/sda7 5598 6814 9775521 83 Linux
```

Keep reading to find out how to *use* this partition. Note that you may need to reboot a system to make new partitions accessible.

Making a filesystem in a partition

Just having a partition is not quite enough; you need to make the filesystem. Above we created a new Linux partition at /dev/sda7, but we need to decide which of the many filesystems Linux supports to use within that partition. Do we want the historical default ext2? Or the newer journaling-enhanced extension ext3 format? Maybe we want one of the enhanced filesystems contributed to Linux by other parties: ReiserFS, XFS, JFS. Or maybe we need a filesystem that interoperates with another operating system, such as Minix, MSDOS, or VFAT (some others can be read if created already, but not always created with Linux tools).

All of the tools for making new filesystems follow the naming convention `mkfs.*`. That is, your system might have `mkfs.ext2`, `mkfs.minix`, `mkfs.xfx`, and so on, usually installed in /sbin/. Also, you may access each of these using the basic `mkfs -t <fstype>` switch. Several, but not all, of the filesystems also have compact forms like `mke3fs`. The filesystems that are available depend on your specific Linux distribution and version, and any extra tools you might have installed. `mkfs.ext2` is available on nearly every distribution.

The basics of making a filesystem are simple. Just run your desired `mkfs.*` tool against the partition you want the filesystem to exist on. For example:

```
% mkfs.xfs /dev/sda7
```

The displayed messages will vary according to the filesystem type you used. Generally, the messages give you information on the number of inodes, blocks, journaling type (if any), extents, and fragments relevant to that particular filesystem's usage strategy. Many of the filesystem creation tools warn you if you try to create a new filesystem on a partition with an existing filesystem, but not all of them will, so proceed with great caution (creating a new filesystem over an old one will probably result in data loss).

Making an ISO filesystem with mkisofs

A special case of making a filesystem is the creation of an *ISO filesystem*, which is a system image that may be written to a writeable CD or DVD device. An ISO filesystem is special in the sense that it is really just a (large) file with data laid out in a certain way rather than in an arrangement of a raw device like /dev/cdrom or /dev/hdb3.

The basic idea of creating an ISO filesystem -- which really means either an ISO9660 or HFS hybrid volume -- is simply to take the files in one or more existing hierarchies and arrange them into an ISO image. ISO9660 itself is limited to simple DOS-style 8.3 names, but the Rock Ridge and Joliet extensions allow storage of longer names and/or additional file attributes. For example, to create an image of a project, you might use a command like:

```
% mkisofs -o ProjectCD.iso -r ~/project-files ~/project-extras
```

In this case, we create an ISO image that uses Rock Ridge attributes (but unlike `-R` sets more useful values, such as all files readable) and contains a merge of all the files in two directories. Other options would let us add bootable headers to the image, create an HFS image, attach directories in specified locations other than root, and fine-tune file options.

Making an ISO filesystem with cdrecord

Transferring an ISO image to a recordable CD or DVD is often accomplished nowadays using a front-end tool, often a GUI interface. For example, both Gnome and KDE make CD burning part of their file-manager interface. Some commercial tools exist also. But for a system administrator, the older command-line tool `cdrecord` is a trusted utility that is present on most modern distributions and is much closer to "standard" than are other front-ends. Generally, the basic usage just requires specifying the device you want to write to and the ISO file you want to write.

As usual with Linux utilities, you may also specify a number of options to the record process, such as `-overburn` for CDs larger than 650 MB or a specific burn speed for your writer. See the manpage for `cdrecord` for current details.

You can find the device with the `-scanbus` option. The device you want is named

as a numeric triple indicating the bus, not a regular block device in the filesystem. For example, you might see something like (abridged):

Listing 3. Finding a recordable device

```
% cdrecord -scanbus
[...]
scsibus0:
0,0,0     0) 'ATA     ' 'WDC WD800UE-00HC' '09.0' Disk
0,1,0     1) *
[...]
scsibus1:
1,0,0   100) 'Slimtype' 'DVDRW SOSW-852S ' 'PSB2' Removable
CD-ROM
[...]
```

With bus information in hand, you can burn an image:

```
% sudo cdrecord -overburn -v speed=16 dev=1,0,0
/media/KNOPPIX_V3.6-2004-08-16-EN.iso
```

In this case, the image is oversized and I know my burner supports 16x. The action command output is rather verbose because of the `-v` option, but that helps in understanding the whole process.

Making an ISO filesystem with dd

Of final note, sometimes you want to create a brand new ISO image not out of some directories in your main filesystem, but rather from an already existing CD or DVD. To make an ISO image from a CD, just use the command `dd`, but refer to the raw block device for the CD rather than to the mounted location:

```
% dd if=/dev/cdrom of=project-cd.iso
```

You might wonder why not just use `cp` if the goal is to copy bytes. Actually, if you ignore a reported I/O error when the raw device runs out of bytes to copy, the `cp` command is *likely* to work. However, `dd` is better style (and doesn't complain, but instead reports a summary of activity).

Section 3. Operating the Linux filesystem

Mounting and unmounting with mount and umount

A flexible feature of Linux systems is the fine-tuned control you have over mounting and unmounting filesystems. Unlike under Windows and some other operating systems, partitions are not automatically assigned locations by the Linux kernel, but

are instead attached to the single / root hierarchy by the `mount` command. Moreover, different filesystem types (on different drives, even) may be mounted within the same hierarchy. You can unmount a particular partition with the `umount` command, specifying either the mount point (such as /home) or the raw device (such as /dev/hda7).

For recovery purposes, the ability to control mount points lets you do forensic analysis on partitions -- using fsck or other tools -- without risk of further damage to a damaged filesystem. You may also custom mount a filesystem using various options; the most important of these is mounting read-only using either of the synonyms `-r` or `-o ro`.

As a quick example, you might want to substitute one user directory location for another, either because of damage to one or simply to expand disk space or move to a faster disk. You might perform this switch using something like:

```
# umount /home # old /dev/hda7 home dir
# mount -t xfs /dev/sda1 /home # new SCSI disk using XFS
# mount -t ext3 /dev/sda2 /tmp # also put the /tmp on SCSI
```

Default mounting

For day-to-day operation, you generally want a pretty fixed set of mounts to happen at every system boot. You control the mounts that happen at bootup by putting configuration lines in the file /etc/fstab. A typical configuration might look something like this:

Listing 4. Sample configuration for mounting at bootup

```
# <file system> <mount point>   <type>  <options>       <dump>  <pass>
       proc            /proc           proc    defaults        0       0
       /dev/sda3       /               reiserfs notail         0       1
       /dev/sda5       none            swap    sw              0       0
       /dev/sda6       /home           ext3    rw              0       2
       /dev/scd0       /media/cdrom0   udf,iso9660 ro,user,noauto  0   0
       /media/Ubuntu-5.04-install-i386.iso /media/Ubuntu_5.04 iso9660
rw,loop 0 0
```

In the above listing, the first field (<file system>) is normally the block device to mount. The second field (<mount point>) is the mounted location. In some special cases, something other than a block device is given first. For `supermount` devices, you will see `none`. `/proc` is another odd case. You might also mount loopback devices, which are usually regular files.

The third field (<type>) and fourth field (<options>) are fairly straightforward; options depend on filesystem type and usage. The fifth field (<dump>) is usually zero. The sixth field (<pass>) should be 1 for the root filesystem and 2 for other filesystems that should be `fsck`ed during system boot.

Automounting with AMD and automount

Linux has quite a few ways of automatically mounting media that is removable (floppies, CDs, USB drives) or otherwise not of fixed availability (such as NFS filesystems). The goal of all these tools is similar, but each works slightly differently.

The tool AMD (automount daemon) is somewhat older and operates in userland. Basically, AMD runs periodically to see if any new mountable filesystems have become available generally for NFS filesystems. For the most part, AMD has been replaced in Linux distributions by Autofs, which runs as a kernel process.

You set up Autofs by compiling it into the kernel you use. After that, the behavior of the Autofs daemon (usually /etc/init.d/autofs) is controlled by the file /etc/auto.master, which in turn references a map file. For example:

```
# Sample auto.master file
# Format of this file: mountpoint map options
/mnt /etc/auto.mnt --timeout=10
```

The referenced `/etc/auto.mnt` specifies one or more subdirectories of /mnt that will be mounted (if access is requested). Unmounting will occur automatically, in this case 10 seconds after last access.

```
# Sample /etc/auto.mnt
floppy -fstype=auto,rw,sync,umask=002 :/dev/fd0
cdrom -fstype=iso9660,ro,nosuid,nodev :/dev/cdrom
remote -fstype=nfs example.com:/some/dir
```

Automounting with supermount and submount

The tools `supermount` and `submount` are kernel-level tools (either compiled into the base kernel or kernel modules) to automatically mount removable media when accessed. `submount` is somewhat newer, but `supermount` is still probably used in more distributions. Neither tool is useful for NFS remote mounts, but either is more seamless than Autofs for local media.

In either case, devices requiring automounting are generally listed in the /etc/fstab configuration. The tools use slightly different syntaxes in /etc/fstab, but both are straightforward. A `supermount`-enabled /etc/fstab might contain the following:

```
# Example of supermount in /etc/fstab
none /mnt/cdrom supermount fs=auto,dev=/dev/cdrom 0 0
none /mnt/floppy supermount fs=auto,dev=/dev/fd0,--,user,rw 0
0
```

`submount` specifies the block device in the regular location rather than as a mount option. For example:

```
/dev/cdrom /mnt/cdrom subfs fs=cdfss,ro,users 0 0
/dev/fd0 /mnt/floppy subfs fs=floppyfss,rw,users 0 0
```

What is currently mounted?

A Linux user has several ways to see a list of current mounts. The `mount` command with no options (or with the `-l` option) lists currently mounted paths. If you like, you can filter the results with the `-t fstype` option.

The underlying dynamic information on mounted filesystems lives in /etc/mtab. The `mount` and `umount` commands and other systems processes will update this file to reflect current status; you should treat this file as read-only. A subset of the `mount` status information is additionally contained in /proc/mounts.

Special tools

The tool `sync` forces changed unwritten blocks to disk. You should not need to use this in normal situations, but you can sometimes check for disk problems by checking for a non-zero exit status. Modern filesystems, particularly journaling filesystems like ext3, Reiser, and JFS, effectively do syncing on every write.

If you like, you can manually disable or enable the use of a swapping or enable/disable swapping for particular devices. Normally, every device marked as type `swap` in /etc/fstab is used for swapping.

Section 4. Maintaining a Linux filesystem

Fixing a filesystem with fsck

Your best friend in repairing a broken filesystem is `fsck`.

The tool called `fsck` is actually just a front-end for a number of more narrow `fsck.*` tools -- `fsck.ext2`, `fsck.ext3`, or `fsck.reiser`. You may specify the type explicitly using the `-t` option, but `fsck` will make an effort to figure it out on its own. Read the manpage for `fsck` or `fsck.*` for more details. The main thing you want to know is that the `-a` option will try to fix everything it can automatically.

You can check an unmounted filesystem by mentioning its raw device. For example, use `fsck /dev/hda8` to check a partition not in use. You can also check a rooted filesystem such as `fsck /home`, but generally do that only if the filesystem is already mounted as read-only, not as read-write.

Checking blocks with badblocks

The `badblocks` utility does a lower-level test of the quality of a block device (or partition) than `fsck` does. `badblocks` may -- destructively or non-destructively -- examine the reliability of blocks on a device by writing and reading test patterns. The default option is `-n` for a slower mode that preserves existing data. For a brand-new partition with no existing files, you can (and probably should) use the `-w`. This tool simply informs you of bad blocks; it does not repair or mark them.

However, in practice, you are usually better off using the badblock-checking wrapper in the `fsck.*` tool for your filesystem. For example, `e2fsck` (also called `fsck.ext2`) has the option `-c` to find and mark badblocks that the `badblocks` tool can detect. ReiserFS has similar `--check` and `--badblocks` options (but is not quite as automatic). Read the documentation for your particular filesystem for details on wrappers to `badblocks`.

Finding other maintenance utilities

Several tools are available for examining and fine-tuning Linux filesystems. In normal usage, the default settings for filesystems are well designed, but occasionally you will want to use filesystem tools for forensic analysis on crashed systems or to tune performance on systems with well-defined usage patterns.

Each filesystem type has its own set of tools; check the documentation for the filesystem you use for more details. Most have a similar array of tools. Some examples include:

- `dumpe2fs`: Output information about an ext2/3 filesystem.
- `tune2fs`: Adjust filesystem parameters on an ext2/3 filesystem.
- `debugfs`: Interactively fine-tune and examine an ext2/3 filesystem.
- `debugreiserfs`: Output information about a Reiser filesystem.
- `reiserfstune`: Adjust filesystem parameters on a Reiser filesystem.
- `xfs_admin`: Adjust filesystem parameters of an XFS filesystem.

Resources

Learn

- At the LPIC Program, find task lists, sample questions, and detailed objectives for the three levels of the Linux Professional Institute's Linux system administration certification.
- "Common threads: Advanced filesystem implementor's guide, Parts 1 - 13" (developerWorks, starting June 2001) is an excellent series on Linux filesystems.
- Find more resources for Linux developers in the developerWorks Linux zone.

Get products and technologies

- Order the SEK for Linux, a two-DVD set containing the latest IBM trial software for Linux from DB2®, Lotus®, Rational®, Tivoli®, and WebSphere®.
- Build your next development project on Linux with IBM trial software, available for download directly from developerWorks.

Discuss

- Participate in the discussion forum for this content.
- Get involved in the developerWorks community by participating in developerWorks blogs.

About the author

David Mertz, Ph.D.
David Mertz is Turing complete, but probably would not pass the Turing Test. For more on his life, see his personal Web page. He's been writing the developerWorks columns *Charming Python* and *XML Matters* since 2000. Check out his book *Text Processing in Python* .